Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №3.1-3.3**

**з курсу: «Інтелектуальні вбудовані системи»**

*Виконав:*

студент групи ІП-84

Шмалько Б. І.

Залікова книжка №8425

*Перевірив:*

Регіда П. Г.

Київ 2020 р.

**Основні теоретичні відомості**

Факторизації лежить в основі стійкості деяких криптоалгоритмів, еліптичних кривих, алгебраїчній теорії чисел та кванових обчислень, саме тому дана задача дуже гостро досліджується, й шукаються шляхи її оптимізації.

На вхід задачі подається число n Є ℕ, яке необхідно факторизувати. Перед виконанням алгоритму слід переконатись в тому, що число не просте. Далі алгоритм шукає перший простий дільник, після чого можна запустити алгоритм заново, для повторної факторизації.

В залежності від складності алгоритми факторизації можна розбити на дві групи:

• Експоненціальні алгоритми (складність залежить експоненційно від довжини вхідного параметру);

• Субекспоненціальні алгоритми.

Метод факторизації Ферма. Ідея алгоритму заключається в пошуку таких чисел А і В, щоб факторизоване число n мало вигляд: n = A 2 – B 2 . Даний метод гарний тим, що реалізується без використання операцій ділення, а лише з операціями додавання й віднімання.

Приклад алгоритму:

Початкова установка: x = [√𝑛] – найменше число, при якому різниця x 2 -n невід’ємна. Для кожного значення k Є ℕ, починаючи з k = 1, обчислюємо (⌈√𝑛⌉ + 𝑘) 2 − 𝑛 і перевіряємо чи не є це число точним квадратом.

• Якщо не є, то k++ і переходимо на наступну ітерацію.

• Якщо є точним квадратом, тобто 𝑥 2 − 𝑛 = (⌈√𝑛⌉ + 𝑘) 2 − 𝑛 = 𝑦 2 , то ми отримуємо розкладання: 𝑛 = 𝑥 2 − 𝑦 2 = (𝑥 + 𝑦)(𝑥 − 𝑦) = 𝐴 ∗ 𝐵, в яких 𝑥 = (⌈√𝑛⌉ + 𝑘)

Якщо воно є тривіальним і єдиним, то n – просте

Важливою задачею яку система реального часу має вирішувати є отримання необхідних для обчислень параметрів, її обробка та виведення результату у встановлений дедлайн. З цього постає проблема отримання водночас точних та швидких результатів. Модель Перцпептрон дозволяє покроково наближати початкові значення.

Розглянемо приклад: дано дві точки А(1,5), В(2,4), поріг спрацювання Р = 4, швидкість навчання δ = 0.1. Початкові значення ваги візьмемо нульовими W1 = 0, W2 = 0. Розрахунок вихідного сигналу y виконується за наступною формулою:

x1 \* W1 + x2 \* W2 = y

Для кожного кроку потрібно застосувати дельта-правило, формула для розрахунку похибки:

Δ = P – y

де y – значення на виході.

Для розрахунку ваги, використовується наступна формули:

W1(i+1) = W1(i) + W2 \* x11

W2(i+1) = W1(i) + W2 \* x12

де i – крок, або ітерація алгоритму.

Генетичні алгоритми служать, головним чином, для пошуку рішень в багатовимірних просторах пошуку.

Можна виділити наступні етапи генетичного алгоритму:

• (Початок циклу)

• Розмноження (схрещування)

• Мутація

• Обчислити значення цільової функції для всіх особин

• Формування нового покоління (селекція)

• Якщо виконуються умови зупинки, то (кінець циклу), інакше (початок циклу)

**Завдання на лабораторну роботу**

Розробити програма для факторизації заданого числа методом Ферма. Реалізувати користувацький інтерфейс з можливістю вводу даних.

Поріг спрацювання: P = 4 Дано точки: А(0,6), В(1,5), С(3,3), D(2,4). Швидкості навчання: δ = {0,001; 0,01; 0,05; 0.1; 0.2; 0,3} Дедлайн: часовий = {0.5с; 1с; 2с; 5с}, кількість ітерацій = {100;200;500;1000} Обрати швидкість навчання та дедлайн. Налаштувати Перцептрон для даних точок. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрати часу та точності результату за різних параметрах навчання.

Налаштувати генетичний алгоритм для знаходження цілих коренів діофантового рівняння ax1+bx2+cx3+dx4=y. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрат часу на розрахунки.

**Лістинг програми**

Лаб 3.1

const Ferma = *(*rawN : string, setResult : React.Dispatch*<*React.SetStateAction*<*string*>>)* => *{* let n = ***Number****(*rawN*)* if *(*isNaN*(*n*)) {* setResult*(*"incorrect input data"*)* return  
 *}* if *(*n % 2 === 0*) {* setResult*(*`first number = 2 and second number = $*{*n / 2*}*`*)* return  
 *}* let b, a = ***Math***.ceil*(****Math***.sqrt*(*n*))* if *(*a \*\* 2 === n*) {* setResult*(*`first number = $*{*a*}* and second number = $*{*a*}*`*)* return  
 *}* while *(*true*) {* let tmp = a \*\* 2 - n;  
 b = ***Math***.ceil*(****Math***.sqrt*(*tmp*))* if *(*b \*\* 2 === tmp*)* break  
 a++  
 *}* setResult*(*`first number = $*{*a - b*}* and second number = $*{*a + b*}*`*)  
}* export default Ferma

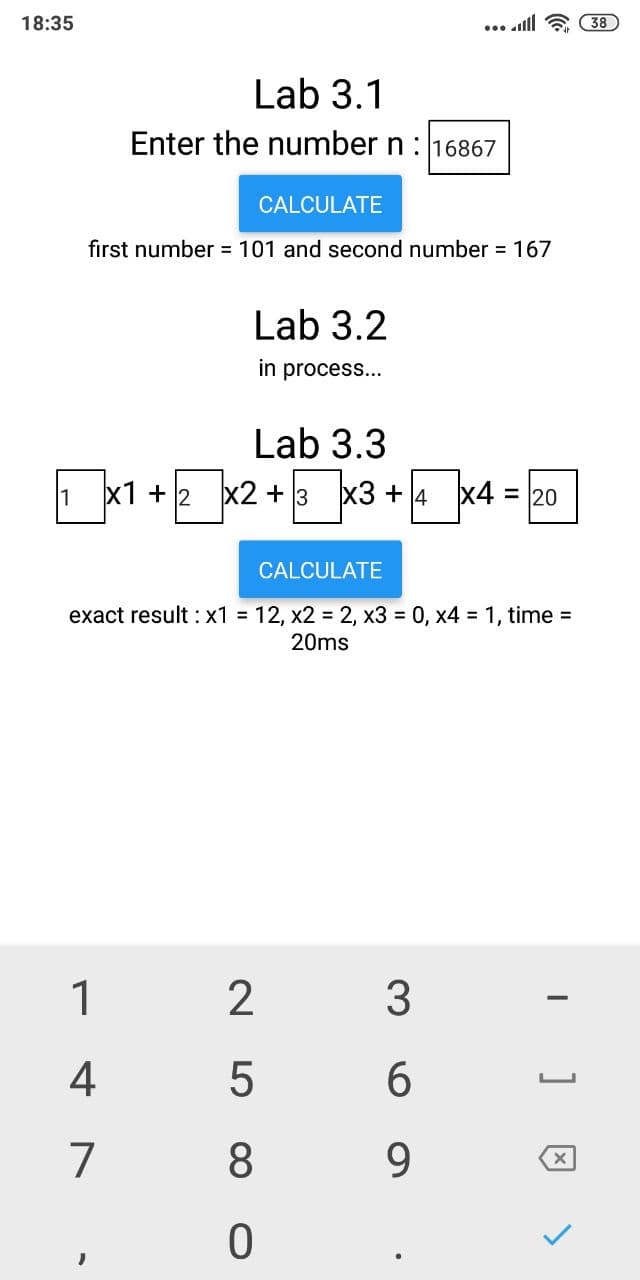
Лаб 3.2

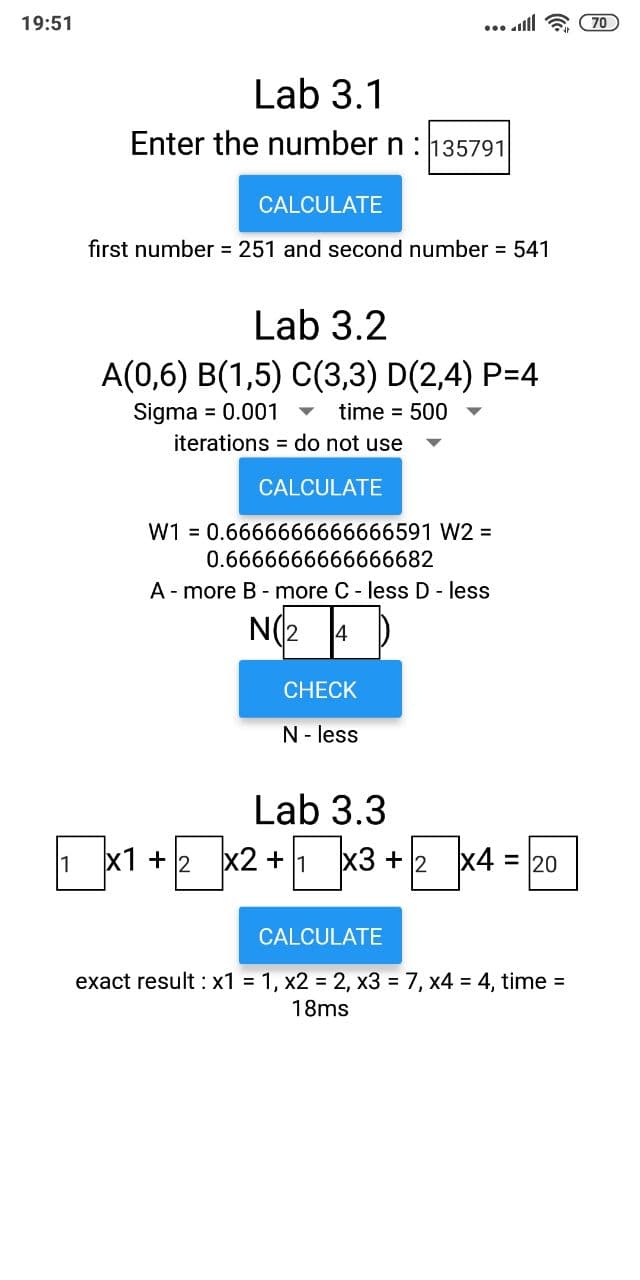
const Perceptron = *(*dots: Array*<*Array*<*number*>>*, P: number, setResult32W: React.Dispatch*<*React.SetStateAction*<*string*>>*, setResult32V: React.Dispatch*<*React.SetStateAction*<*string*>>*, rowSigma: string, rowTime: string, rowIterations : string*)* => *{* let Ws : Array*<*number*>* = *[*0,0*]*,  
 sigma = ***Number****(*rowSigma*)*,  
 time = ***Number****(*rowTime*)*,  
 iterations = ***Number****(*rowIterations*)* if*(*isNaN*(*time*)* && isNaN*(*iterations*)) {* setResult32W*(*'incorrect data (select iterations or time)'*)* return Ws  
 *}* let delta = 0  
  
  
 const calculateDelta = *(*dotNum : number*)* => *{* delta = P - *(*Ws*[*0*]*\*dots*[*dotNum*][*0*]* + Ws*[*1*]*\*dots*[*dotNum*][*1*])* if *(*delta <= 0*)* return 'more'  
 return 'less'  
 *}* const calculateWs = *(*dotNum : number*)* => *{* Ws*[*0*]* = Ws*[*0*]* + delta\*dots*[*dotNum*][*0*]*\*sigma  
 Ws*[*1*]* = Ws*[*1*]* + delta\*dots*[*dotNum*][*1*]*\*sigma  
 *}* let start = new ***Date****()*.getTime*()*,  
 currentIteration = 0  
 while *(*new ***Date****()*.getTime*()* - start < time || currentIteration < iterations*){* calculateDelta*(*currentIteration % 4*)* calculateWs*(*currentIteration % 4*)* currentIteration++  
 *}* setResult32W*(*`W1 = $*{*Ws*[*0*]}* W2 = $*{*Ws*[*1*]}*`*)* setResult32V*(*`A - $*{*calculateDelta*(*0*)}* B - $*{*calculateDelta*(*1*)}* C - $*{*calculateDelta*(*2*)}* D - $*{*calculateDelta*(*3*)}*`*)* return Ws  
*}*export default Perceptron

Лаб 3.3

type resultType = *{* x1 : number  
 x2 : number  
 x3 : number  
 x4 : number  
 fitness ?: number  
 time ?: number  
*}*type genType = *{* gen : Array*<*number*>* fitness : number  
 likelihood : number  
*}*type populationType = Array*<*genType*>*const geneticAlgoritm = *(*a: number, b: number, c: number, d: number, y: number*)* => *{* let result : resultType = *{* x1 : 0,  
 x2 : 0,  
 x3 : 0,  
 x4 : 0,  
 time : 0  
 *}* const MAX\_POPULATION = 300  
 const ITERATIONS = 200  
 let population : populationType = *[]* const calculateFitness = *()* => *{* for*(*let i = 0; i < MAX\_POPULATION; i++*) {* let total = a \* population*[*i*]*.gen*[*0*]* + b \* population*[*i*]*.gen*[*1*]* + c \* population*[*i*]*.gen*[*2*]* + d \* population*[*i*]*.gen*[*3*]*;  
 population*[*i*]*.fitness = ***Math***.abs*(*total - y*)* if *(*population*[*i*]*.fitness === 0*) {* return i;  
 *}  
 }* return -1;  
 *}* const chooseBest = *()* => *{* let best = ***Infinity*** let who = 0;  
 for*(*let i = 0; i < MAX\_POPULATION; i++*) {* if*(*population*[*i*]*.fitness < best*) {* best = population*[*i*]*.fitness  
 who = i  
 *}  
 }* return who  
 *}* const calculateLikelihoods = *()* => *{* let total = 0  
 for*(*let i = 0; i < MAX\_POPULATION; i++*) {* total += 1/population*[*i*]*.fitness  
 *}* for*(*let i = 0; i < MAX\_POPULATION; i++*) {* population*[*i*]*.likelihood = *(*1/population*[*i*]*.fitness*)*/total  
 *}  
 }* const chooseParent = *()* => *{* const allChances = *[]* for*(*let i = 0; i < MAX\_POPULATION; i++*) {* allChances*[*i*]* = ***Math***.floor*(****Math***.random*()* \* population*[*i*]*.fitness \* 1000*)  
 }* let best = -***Infinity*** let who = 0;  
 for*(*let i = 0; i < MAX\_POPULATION; i++*) {* if*(*allChances*[*i*]* > best*) {* best = allChances*[*i*]* who = i  
 *}  
 }* return who  
 *}* const generateChildren = *()* => *{* const populationChildren : populationType = *[]* for*(*let i = 0; i < MAX\_POPULATION; i++*) {* let dad = chooseParent*()* let mum = chooseParent*()* populationChildren*[*i*]* = *{*gen : *[]*, fitness : ***Infinity***, likelihood : 0*}* populationChildren*[*i*]*.gen = *[* population*[*dad*]*.gen*[*0*]*,  
 population*[*dad*]*.gen*[*1*]*,  
 population*[*mum*]*.gen*[*2*]*,  
 population*[*mum*]*.gen*[*3*]*,  
 *]  
 }* population = populationChildren  
 *}* let start = new ***Date****()*.getTime*()* let fitness  
  
 for*(*let i = 0; i < MAX\_POPULATION; i++*) {* population*[*i*]* = *{*gen : *[]*, fitness : ***Infinity***, likelihood : 0*}* for *(*let j = 0; j < 4; j++*) {* population*[*i*]*.gen*[*j*]* = ***Math***.floor*(****Math***.random*()* \* y*)  
 }  
 }* fitness = calculateFitness*()* ***console***.log*(*fitness*)* if *(*fitness !== -1*) {* let end = new ***Date****()*.getTime*()* result.x1 = population*[*fitness*]*.gen*[*0*]* result.x2 = population*[*fitness*]*.gen*[*1*]* result.x3 = population*[*fitness*]*.gen*[*2*]* result.x4 = population*[*fitness*]*.gen*[*3*]* result.time = end - start  
 return result;  
 *}* let iterations = 0  
 while *(*iterations < ITERATIONS*) {* calculateLikelihoods*()* generateChildren*()* fitness = calculateFitness*()* if *(*fitness !== -1*) {* let end = new ***Date****()*.getTime*()* result.x1 = population*[*fitness*]*.gen*[*0*]* result.x2 = population*[*fitness*]*.gen*[*1*]* result.x3 = population*[*fitness*]*.gen*[*2*]* result.x4 = population*[*fitness*]*.gen*[*3*]* result.time = end - start  
 return result;  
 *}*;  
 iterations++;  
 *}* let best = chooseBest*()* let end = new ***Date****()*.getTime*()* result.x1 = population*[*best*]*.gen*[*0*]* result.x2 = population*[*best*]*.gen*[*1*]* result.x3 = population*[*best*]*.gen*[*2*]* result.x4 = population*[*best*]*.gen*[*3*]* result.fitness = population*[*best*]*.fitness  
 result.time = end - start  
 return result;  
*}*const Genetic = *(*rawA: string, rawB: string, rawC: string, rawD: string, rawY: string, setResult : React.Dispatch*<*React.SetStateAction*<*string*>>)* => *{* const a = ***Number****(*rawA*)*,  
 b = ***Number****(*rawB*)*,  
 c = ***Number****(*rawC*)*,  
 d = ***Number****(*rawD*)*,  
 y = ***Number****(*rawY*)* if *(*isNaN*(*a*)* || isNaN*(*b*)* || isNaN*(*c*)* || isNaN*(*d*)* || isNaN*(*y*)) {* setResult*(*"incorrect input data"*)* return  
 *}* const res = geneticAlgoritm*(*a,b,c,d,y*)* if *(*res.fitness !== 0 && res.fitness*){* setResult*(*`not accurate result : x1 = $*{*res.x1*}*, x2 = $*{*res.x2*}*, x3 = $*{*res.x3*}*, x4 = $*{*res.x4*}*, time = $*{*res.time*}*ms, fitness = $*{*res.fitness*}*`*)* return  
 *}* setResult*(*`exact result : x1 = $*{*res.x1*}*, x2 = $*{*res.x2*}*, x3 = $*{*res.x3*}*, x4 = $*{*res.x4*}*, time = $*{*res.time*}*ms`*)  
}*export default Genetic

**Результати роботи програми**

****

****

**Висновки**

В даній лабораторній роботі ми – ознайомитись з основними принципами розкладання числа на прості множники з використанням різних алгоритмів факторизації, ознайомилися з принципами машинного навчання за допомогою математичної моделі сприйняття інформації Перцептрон (Perceptron) а також ознайомилися з принципами реалізації генетичного алгоритму, вивчили та дослідили особливостей даного алгоритму з використанням засобів моделювання і сучасних програмних оболонок. На основі знайдених даних створили інтерфейс (повноцінний додаток) для користувача мобільного телефону. Визначили час роботи даних алгоритмів. Дані знання знадобляться нам у нашій подальшій кар’єрі.